Inherent-Cost Aware Collective SpatialKeyword Queries

Inherent-Cost Aware Collective Spatial Keyword Queries

Harry Kai-Ho Chan[1], Cheng Long[2], and Raymond Chi-Wing Wong[1]

[1] The Hong Kong University of Science and Technology
{khchanak, raywong}@cse.ust.hk
[2] Queen's University Belfast
cheng.long@qub.ac.uk

Abstract. With the proliferation of spatial-textual data such as location-based services and geo-tagged websites, spatial keyword queries become popular in the literature. One example of these queries is the *collective spatial keyword query* (CoSKQ) which is to find a set of objects in the database such that it *covers* a given set of query keywords collectively and has the smallest *cost*. Some existing cost functions were proposed in the literature, which capture different aspects of the distances among the objects in the set and the query. However, we observe that in some applications, each object has an inherent cost (e.g., workers have monetary costs) which are not captured by any of the existing cost functions. Motivated by this, in this paper, we propose a new cost function called the *maximum dot size cost* which captures both the distances among objects in a set and a query as existing cost functions do and the inherent costs of the objects. We prove that the CoSKQ problem with the new cost function is NP-hard and develop two algorithms for the problem. One is an exact algorithm which is based on a novel search strategy and employs a few pruning techniques and the other is an approximate algorithm which provides a $\ln |q.\psi|$ approximation factor, where $|q.\psi|$ denotes the number of query keywords. We conducted extensive experiments based on both real datasets and synthetic datasets, which verified our theoretical results and efficiency of our algorithms.

1 Introduction

Nowadays, geo-textual data which refers to data with both spatial and textual information is ubiquitous. Some examples of geo-textual data include the spatial points of interest with textual description, geo-tagged web objects (e.g., webpages and photos at Flicker), and also geo-social networking data (e.g., users of FourSquare have their check-in histories which are spatial and also profiles which are textual).

Collective Spatial Keyword Query (CoSKQ) [3,18,2] is a query type recently proposed on geo-textual data which is described as follows. Let $\mathcal{O}$ be a set of objects. Each object $o \in \mathcal{O}$ is associated with a spatial location, denoted by $o.\lambda$, and a set of keywords, denoted by $o.\psi$. Given a query q with a location $q.\lambda$ and a set of keywords $q.\psi$, CoSKQ is to find a set S of objects such that S covers $q.\psi$, i.e., $q.\psi \subseteq \cup_{o \in S} o.\psi$, and the $cost$ of S, denoted by $cost(S)$, is minimized. CoSKQ is useful in applications where a user wants to find a set of objects to collectively satisfy his/her needs. One example is that a tourist wants to find some points-of-interest to do sight-seeing, shopping and dining, where the user's needs could be captured by the query keywords of a CoSKQ

query. Another example is that a manager wants to finds some workers, collectively offering a set of skills, to set up a project.

One key component of the CoSKQ problem is the cost function that measures the cost of a set of objects S wrt the query q, i.e., $cost(S)$. In the literature, five different cost functions have been proposed for $cost(S)$ [3,18,2]. Each of these cost functions is based on one or a few of the following distances: (D1) the *sum* of the distances between the objects in S and q, (D2) the *min* of the distances between the objects in S and q, (D3) the *max* of the distances between the objects in S and q, and (D4) the *max* of the distances between two objects in S. Specifically, (1) the *sum cost function* [3,2] defines $cost(S)$ as D1, (2) the *maximum sum cost function* [3,18,2] defines $cost(S)$ as D3 + D4, (3) the *diameter cost function* [18] defines $cost(S)$ as $\max\{\text{D1, D2}\}$, (4) the *sum max cost function* [2] defines $cost(S)$ as D1 + D4, and (5) the *min max cost function* [2] defines $cost(S)$ as D2 + D4.

These cost functions are suitable in applications where the cost of a set of objects could be captured well by the spatial distances among the objects and the query *only*. For example, in the application where a tourist wants to find a set of points-of-interest, the cost is due to the distances to travel among the points-of-interest and the query. However, in some other applications, each object has an inherent cost and thus the cost of a set of objects would be better captured by both the distances among the objects and the query and the inherent costs of the objects. In the application where a manager wants to find a group of workers, each worker is associated with some monetary cost. Another example is that a tourist wants to visit some POIs (e.g, museums, parks), each POI is associated with an admission fee. Motivated by this, in this paper, we propose a new cost function called *maximum dot size function* which captures both some spatial distances between objects and a query and the inherent costs of the objects. Specifically, the *maximum dot size function* defines the cost of a set S of objects, denoted by $cost_{MaxDotSize}(S)$, as the multiplication between the maximum distance between the objects in S and q and the sum of the inherent costs of objects in S.

The CoSKQ problem with the maximum dot size function is proven to be NP-hard and an exact algorithm would run in exponential time where the exponent is equal to the number of query keywords. We design an exact algorithm called *MaxDotSize-E*, which adopts a novel strategy of traversing possible sets of objects so that only a small fraction of the search space is traversed (which is achieved by designing effective pruning techniques which are made possible due to the search strategy). For better efficiency, we also design an approximate algorithm called *MaxDotSize-A* for the problem. Specifically, our main contribution is summarized as follows.

- Firstly, we propose a new cost function $cost_{MaxDotSize}$, which captures both the spatial distances between the objects and the query, and the inherent costs of the objects.
- Secondly, we prove the NP-hardness of *MaxDotSize-CoSKQ* and design two algorithms, namely *MaxDotSize-E* and *MaxDotSize-A*. *MaxDotSize-E* is an exact algorithm and runs faster than an adapted algorithm in the literature [3]. *MaxDotSize-A* gives a solution set S with a $\ln |q.\psi|$-factor approximation. In particular, if $|S| \leq 3$ it is guaranteed that S is an optimal solution.

– Thirdly, we conducted extensive experiments on both real and synthetic datasets, which verified our theoretical results and the efficiency of our algorithms.

The rest of this paper is organized as follows. Section 2 gives the related work. Section 3 defines the problem and discusses its hardness. Section 4 presents our proposed algorithms. Section 5 gives the empirical study. Section 6 concludes the paper.

2 Related Work

Many existing studies on spatial keyword queries focus on retrieving a *single object* that is close to the query location and relevant to the query keywords.

A *boolean kNN query* [13,5,25,30] finds a list of k objects each covering all specified query keywords. The objects in the list are ranked based on their spatial proximity to the query location.

A *top-k kNN query* [9,19,16,20,21,10,26] adopts the ranking function considering both the spatial proximity and the textual relevance of the objects and returns top-k objects based on the ranking function. This type of queries has been studied on Euclidean space [9,19,16], road network databases [20], trajectory databases [21,10] and moving object databases [26]. Usually, the methods for this kind of queries adopt an index structure called the *IR-tree* [9,24] capturing both the spatial proximity and the textual information of the objects to speed up the keyword-based nearest neighbor (NN) queries and range queries. In this paper, we also adopt the *IR-tree* as an index structure.

Some other studies on spatial keyword queries focus on finding an *object set* as a solution. Among them, some [3,18,2] studied the *collective spatial keyword queries* (CoSKQ). These studies on CoSKQ adopted a few cost functions which capture the spatial distances among the objects and the query only and thus they are not suitable in applications where objects are associated with inherent costs.

Another query that is similar to CoSKQ is the *mCK query* [28,29,15] which takes a set of m keywords as input and finds m objects with the minimum *diameter* that cover the m keywords specified in the query. In the existing studies of mCK queries, it is usually assumed that each object contains a single keyword. There are some variants of the mCK query, including the *SK-COVER* [8] and the *BKC query* [11]. These queries are similar to the CoSKQ problem in that they also return an object set that covers the query keywords, but they only take a set of keywords as input.

There are also some other studies on spatial keyword queries, including [22] which finds top-k groups of objects with the ranking function considering the spatial proximity and textual relevance of the groups, [17] which takes a set of keywords and a clue as inputs, and returns k objects with highest similarities against the clue, [14,23] which finds an object set in the road network, [4,12] which finds a *region* as a solution and [1,27] which finds a *route* as a solution.

3 Problem Definition

Let $\mathcal{O}$ be a set of objects. Each object $o \in \mathcal{O}$ is associated with a location denoted by $o.\lambda$, a set of keywords denoted by $o.\psi$ and an inherent cost denoted by $o.cost$. Given

two objects o and o', we denote by $d(o, o')$ the Euclidean distance between $o.\lambda$ and $o'.\lambda$. Given a query q which consists of a location $q.\lambda$ and a set of keywords $q.\psi$, an object is said to be **relevant** if it contains at least one keyword in $q.\psi$, and we denote by $\mathcal{O}_q$ the set containing all relevant objects and say a set of objects is **feasible** if it covers $q.\psi$.

Problem Definition [3]. Given a query $q = (q.\lambda, q.\psi)$, the *Collective Spatial Keyword Query* (CoSKQ) problem is to find a set S of objects in $\mathcal{O}$ such that S covers $q.\psi$ and the *cost* of S is minimized.

In this paper, we propose a new cost function called *maximum dot size* which defines the cost of a set S of objects, denoted by $cost_{MaxDotSize}(S)$, as the multiplication of the maximum distance between objects in S and q and the sum of the inherent costs of objects in S, i.e.,

$$cost_{MaxDotSize}(S) = \max_{o \in S} d(o, q) \cdot \sum_{o \in S} o.cost \tag{1}$$

For simplicity, we assume that each object has a unit cost and thus the overall inherent cost of a set of objects corresponds to the size of this set and $cost_{MaxDotSize}(S)$ corresponds to $\max_{o \in S} d(o, q) \cdot |S|$. However, the exact algorithm and the approximate algorithm developed in this paper could also be applied to the general case with arbitrary costs, while all theoretical results (e.g., approximation ratio) remain applicable (details could be found in Section 4.4).

We define the maximum dot size function with distance D3 (i.e., $\max_{o \in S} d(o, q)$) because D3 is the distance traveled to the most far-away object, and it is able to capture other distances (e.g., $2 \max_{o \in S} d(o, q) \leq \max_{o_1, o_2 \in S} d(o_1, o_2)$). We use a simple product to combine the two factors (distance and inherent cost) such that it remains applicable and meaningful when the object inherent costs are unavailable. Note that we discarded the normalization term $\max_{o \in \mathcal{O}} d(o, q) \cdot |q.\psi| \cdot \max_{o \in \mathcal{O}} o.cost$, which does not affect the applicability of the cost function.

The CoSKQ problem with the maximum dot size function is denoted as **MaxDotSize-CoSKQ**. In the following, if there is no ambiguity, we write $cost_{MaxDotSize}(\cdot)$ as $cost(\cdot)$ for simplicity.

Intractability. The following lemma shows the NP-hardness of MaxDotSize-CoSKQ.

Lemma 1. *MaxDotSize-CoSKQ is NP-hard.* □

Proof: We prove by transforming the *set cover* problem which is known to be NP-Complete to the MaxDotSize-CoSKQ problem. The description of the MaxDotSize-CoSKQ problem is given as follows. Given a set $\mathcal{O}$ of spatial objects, each $o \in \mathcal{O}$ is associated with a location $o.\lambda$ and a set of keywords $o.\psi$, a query q consisting of a query location $q.\lambda$ and a set of query keywords $q.\psi$, and a real number C, the problem is to determine whether there exists a set S of objects in $\mathcal{O}$ such that S covers the query keywords and $cost(S)$ is at most C.

The description of the set cover problem is given as follows. Given a universe set $U = \{e_1, e_2, ..., e_n\}$ of n elements and a collection of m subsets of U, $V = \{V_1, V_2, ..., V_m\}$, and a number k, the problem is to determine whether there exists a set $T \subseteq V$ such that T covers all the elements in U and $|T| \leq k$.

We transform a set cover problem instance to a MaxDotSize-CoSKQ problem instance as follows. We construct a query q by setting $q.\lambda$ to be an arbitrary location in the space and $q.\psi$ to be a set of n keywords each corresponding to an element in U. We construct a set $\mathcal{O}$ such that $\mathcal{O}$ contains m objects each corresponding to a subset in V. For each object o in $\mathcal{O}$, we set $o.\lambda$ to be any location at the boundary of the disk which is centered at $q.\lambda$ and has the radius equal to 1, set $o.\psi$ be a set of keywords corresponding to the elements in the subset in V that o is corresponding to and set $o.cost = 1$. Note that for any $o \in \mathcal{O}$, we have $d(o, q) = 1$ and the MaxDotSize cost of any set of objects is exactly equal to the size of the set. Besides, we set C to be equal to k. Clearly, the above transformation can be done in polynomial time.

The equivalence between the set cover problem instance and the corresponding MaxDotSize-CoSKQ problem instance could be verified easily since if there exists a set T such that T cover all the elements in U and $|T| \leq k$, the set of objects corresponding the subsets in T cover all the query keywords in $q.\psi$ and has the MaxDotSize cost at most C and vice versa. □

4 Algorithms for MaxDotSize-CoSKQ

4.1 An Exact Algorithm

In this section, we present our exact method called *MaxDotSize-E* for MaxDotSize-CoSKQ. Before we present the algorithm, we first introduce some notations. Given a query q and a non-negative real number r, we denote the circle or the disk centered at $q.\lambda$ with radius r by $D(q, r)$. Given a feasible set S, the **query distance owner** of S is defined to be the object $o \in S$ that is most far away from $q.\lambda$ (i.e., $o = \arg\max_{o \in S} d(o, q)$). Given a query q and a keyword t, the **t-keyword nearest neighbor** of q, denoted by $NN(q, t)$, is defined to be the nearest neighbor (NN) of q containing keyword t. Besides, we define the **nearest neighbor set** of q, denoted by $N(q)$, to be the set containing q's t-keyword nearest neighbor for each $t \in q.\psi$, i.e., $N(q) = \cup_{t \in q.\psi} NN(q, t)$. Note that $N(q)$ is a feasible set.

The *MaxDotSize-E* algorithm is presented in Algorithm 1. At the beginning, it maintains an object set S for storing the best-known solution found so far, which is initialized to $N(q)$. Then, it performs an iterative process where each iteration involves three steps.

1. **Step 1 (Query Distance Owner Finding):** It picks one relevant object o following an ascending order of $d(o, q)$.
2. **Step 2 (Feasible Set Construction):** It constructs the best feasible set S' with object o as the query distance owner via a procedure called "findBestFeasibleSet".
3. **Step 3 (Optimal Set Updating):** It then updates S to S' if $cost(S') < cost(S)$.

The iterative process continues with the next relevant object until all relevant objects have been processed.

One remaining issue in Algorithm 1 is the "findBestFeasibleSet" procedure which is shown in Algorithm 2.

First, it maintains a variable ψ, denoting the set of keywords in $q.\psi$ not covered by o yet, which is initialized as $q.\psi - o.\psi$. If $\psi = \emptyset$, then it returns $\{o\}$ immediately since we know that it is the best feasible set. Otherwise, it proceeds to retrieve the

Algorithm 1 Algorithm *MaxDotSize-E*

Input: A query q and a set $\mathcal{O}$ of objects
Output: A feasible set S with the smallest cost

$S \leftarrow N(q)$
// Step 1 (Query Distance Owner Finding)
for each relevant object o in ascending order of $d(o, q)$ **do**
    // Step 2 (Feasible Set Construction)
    $S' \leftarrow$ findBestFeasibleSet(o)
    // Step 3 (Optimal Set Updating)
    if $S' \neq \emptyset$ and $cost(S') < cost(S)$ **then**
        $S \leftarrow S'$
return S

Algorithm 2 Algorithm for finding the best feasible set with object o as the query distance owner (findBestFeasibleSet(o))

Input: An object o
Output: The best feasible set with o as the query distance owner (if any)

$\psi \leftarrow q.\psi - o.\psi$
if $\psi = \emptyset$ **then**
    return $\{o\}$
$\mathcal{O}' \leftarrow$ a set of all relevant objects in $D(q, d(o, q))$
if $\mathcal{O}'$ does not cover ψ **then**
    return $\emptyset$
for each subset S'' of $\mathcal{O}'$ with size at most $\min\{|\psi|, \frac{cost(S)}{d(o,q)} - 1\}$ in ascending order of $|S''|$ **do**
    if S'' covers ψ **then**
        return $S'' \cup \{o\}$
return $\emptyset$

set $\mathcal{O}'$ of all relevant objects in $D(q, d(o, q))$. If $\mathcal{O}'$ does not cover ψ, it returns $\emptyset$ immediately. Otherwise, it enumerates each possible subset S'' of $\mathcal{O}'$ with size at most $\min\{|\psi|, \frac{cost(S)}{d(o,q)} - 1\}$ in ascending order of $|S''|$ (by utilizing the inverted lists maintained for each keyword in ψ), and checks whether S'' covers ψ (note that $|S''|$ is at most $\frac{cost(S)}{d(o,q)} - 1$ because otherwise it cannot contribute to a better solution). If yes, it returns $S'' \cup \{o\}$ immediately since it is the best feasible set in $\mathcal{O}'$. Otherwise, it checks the next subset of $\mathcal{O}'$. When all subsets of $\mathcal{O}'$ have been processed and there is no feasible set found, it returns $\emptyset$.

To further improve the efficiency of the *MaxDotSize-E* algorithm, we develop some pruning techniques in both Step 1 and Step 2.

4.1.1 Pruning in Step 1

The major idea is that not each object $o \in \mathcal{O}_q$ is necessary to be considered as a query distance owner of S' to be constructed and thus some can be pruned. Specifically, we have the following lemmas.

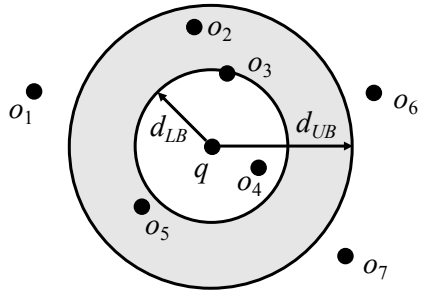


Fig. 1. Distance constraint for query distance owners

Lemma 2 (Distance Constraint). *Let S_o be the optimal set and o be the query distance owner of S_o. Then, we have $d_{LB} \leq d(o,q) \leq d_{UB}$, where $d_{LB} = \max_{o \in N(q)} d(o,q)$ and $d_{UB} = cost(S)$, where S is an arbitrary feasible set.* □

Proof: First, we prove $d_{LB} \leq d(o,q)$ by contradiction. Assume $d(o,q) < d_{LB}$. Let o_f be the farthest object from q in $N(q)$, i.e., $d_{LB} = d(q,o_f)$. There exists a keyword $t_f \in o_f.\psi \cap q.\psi$ such that t_f is not contained by any object that is closer to q than o_f since otherwise $o_f \notin N(q)$. This leads to a contradiction since there exists an object $o' \in S_o$ which covers t_f and $d(o',q) \leq d(o,q) < d_{LB}$.

Second, we prove $d(o,q) \leq d_{UB}$ also by contradiction. Assume $d(o,q) > d_{UB}$. Then $cost(S_o) = d(o,q) \cdot |S_o| > d_{UB} \cdot |S_o| > cost(S)$ which contradict the fact that S_o is the optimal set. □

Figure 1 shows the distance constraint. We only need to consider the relevant objects inside the gray area (i.e., o_2, o_3 and o_5) to be the query distance owners.

Lemma 3 (Keyword Constraint). *Let o be the query distance owner of the set S' to be constructed. If $d(o,q) > d_{LB}$ and $|o.\psi \cap q.\psi| < 2$, there exist a feasible set S'' s.t. $cost(S'') \leq cost(S')$.* □

Proof: Given the set S', we can construct the feasible set S'' as follows. Consider the following two cases. *Case 1.* $d(o,q) > d_{LB}$ and $|o.\psi \cap q.\psi| = 0$. In this case, we can construct $S'' = S' \setminus \{o\}$ be the feasible set with lower cost. *Case 2.* $d(o,q) > d_{LB}$ and $|o.\psi \cap q.\psi| = 1$. Let the keyword $t = o.\psi \cap q.\psi$. We know that $o \notin N(q)$ because $d(o,q) > d_{LB}$. Note that there exist an object $o' \in N(q)$ that contains t. Thus, we can construct $S'' = S' \setminus \{o\} \cup \{o'\}$ be the feasible set with $cost(S'') \leq cost(S')$. □

The above two lemmas suggest that an object o could be pruned if any of the following two conditions is not satisfied.

1. (Condition 1): $d_{LB} \leq d(o,q) \leq d_{UB}$; and
2. (Condition 2): $(d(o,q) = d_{LB})$ or $(d(o,q) > d_{LB}$ and $|o.\psi \cap q.\psi| \geq 2)$

4.1.2 Pruning in Step 2

The major idea is that not each object $o' \in \mathcal{O}'$ is necessary to be enumerated when finding the best feasible set S' with the query distance owner o. We first introduce a concept called **dominance**. Given a set of keywords ψ, a set of objects $\mathcal{O}'$, two objects o_1 and o_2 in $\mathcal{O}'$, we say that o_1 dominates o_2 wrt ψ if all keywords in ψ that are covered by o_2 can be covered by o_1 and there exists a keyword in ψ that is covered by o_1 but not by o_2 (i.e., $o_2.\psi \cap \psi \subset o_1.\psi \cap \psi$). An object that is not dominated by another object is said to be a **dominant** object. Then, it could be verified easily that only those objects that are dominant ones need to be considered in Step 2.

4.1.3 Correctness and Time Complexity

Based on the description of the *MaxDotSize-E* algorithm, it is easy to verify that *MaxDotSize-E* is an exact algorithm.

Theorem 1. *MaxDotSize-E returns a feasible set with the smallest cost for MaxDotSize-CoSKQ.* □

Time Complexity. We use the IR-tree built on $\mathcal{O}$ to support the range query operations involved in the algorithm.

Let n_1 be the number of iterations (lines 3-8) *MaxDotSize-E* and β be the cost of executing one iteration. The time complexity of *MaxDotSize-E* is $O(n_1 \cdot \beta)$. In practice, we have $n_1 << |\mathcal{O}_q|$ since n_1 is equal to the number of relevant objects satisfying the two conditions.

Consider β. It is dominated by the cost of executing the "findBestFeasibleSet" procedure with Algorithm 2 (line 5 of Algorithm 1). We analyze the cost of Algorithm 2 as follows. The cost of lines 1-3 is dominated by that of the remaining parts in the algorithm. Line 4 could be finished by performing a range query with an additional constraint that the object is relevant, which incurs the cost of $O(\log|\mathcal{O}| + |\mathcal{O}'|)$ [7]. Note that $|\mathcal{O}'|$ corresponds to the number of objects returned by the range query. Lines 5-6 could be finished simply by traversing for each object in $\mathcal{O}'$ the set of keywords associated with it and thus the cost is $O(\sum_{o\in\mathcal{O}'}|o.\psi|)$. Lines 7-9 could be finished by enumerating all possible subsets of $\mathcal{O}'$ with size at most $\min\{|\psi|, \frac{cost(S)}{d(o,q)} - 1\}$ in ascending order of size and for each subset, checking whether it covers ψ, and thus the cost is bounded by $O(|\mathcal{O}'|^{|\psi|}\sum_{o\in\mathcal{O}'}|o.\psi|)$ (since there are at most $|\mathcal{O}'|^{|\psi|}$ subsets to be checked and the cost of checking each subset is bounded by $\sum_{o\in\mathcal{O}'}|o.\psi|$). Overall, we know that the time complexity of *MaxDotSize-E* is $O(n_1 \cdot (\log|\mathcal{O}| + |\mathcal{O}'| + \sum_{o\in\mathcal{O}'}|o.\psi| + |\mathcal{O}'|^{|\psi|}\sum_{o\in\mathcal{O}'}|o.\psi|)) = O(n_1 \cdot (\log|\mathcal{O}| + |\mathcal{O}'|^{|\psi|}\sum_{o\in\mathcal{O}'}|o.\psi|))$, where we have $n_1 << |\mathcal{O}_q|$, $|\mathcal{O}'| \leq |\mathcal{O}_q|$, and $|\psi| < |q.\psi|$.

4.2 An Approximate Algorithm

In this section, we introduce an approximate algorithm called *MaxDotSize-A* for MaxDotSize-CoSKQ, which gives a $\ln|q.\psi|$-factor approximation.

MaxDotSize-A is exactly the same as the *MaxDotSize-E* except that it replaces the "findBestFeasibleSet" procedure which is an expensive exhaustive search process with

a greedy process which is much cheaper. Specifically, the greedy process first initializes the set to be returned to $\{o\}$ and then iteratively selects an object in the disk $D(q, d(o, q))$ which has the greatest number of uncovered keywords and inserts it into the set until all keywords are covered.

Theoretical Analysis. Although the set S returned by the *MaxDotSize-A* algorithm might have a larger cost than the optimal set S_o, the difference is bounded.

Theorem 2. *MaxDotSize-A gives a* $\ln |q.\psi|$*-factor approximation for the MaxDotSize-CoSKQ problem. In particular, if the solution returned by MaxDotSize-A has the size at most 3, the solution is an exact solution.* □

Proof: Let S_o be the optimal solution and S_a be the solution returned by *MaxDotSize-A*. Let o be the query distance owner of S_o. There exists an iteration that the algorithm processes o and the greedy process construct the set S' by selects the objects in $D(q, d(o, q))$.

In the following, we show that $cost(S') \leq \ln |q.\psi| \cdot cost(S_o)$ which immediately implies the correctness of the theorem since $cost(S_a) \leq cost(S')$.

Let $S' = \{o, o'_1, o'_2, ..., o'_a\}$ and $S_o = \{o, o_1, o_2, ..., o_b\}$. That is, $|S'| = 1 + a$ and $|S_o| = 1 + b$. Without loss of generality, assume that after o is included in S', $o'_1, o'_2, ..., o'_a$ are included in S' sequentially in the greedy procedure.

Let ψ_t denote the set of keywords in $q.\psi$ that are not covered by S' after o'_{t-1} (if any) is included in S' but before o'_t is included in S' for $t = 1, 2, ..., a$. For example, $\psi_1 = q.\psi - o.\psi$.

In the case that $|S'| = 1$, i.e., $S' = \{o\}$ and $a = 0$, we know that o covers all the keywords in $q.\psi$ and $S_o = \{o\} = S'$. Therefore, S' is an exact solution. In the case that $|S'| = 2$, i.e., $S' = \{o, o'_1\}$ and $a = 1$, S_o also involves exactly two objects and thus $cost(S_o) = d(o, q) \cdot 2 = cost(S')$. Again, S' is an exact solution. In the case that $|S'| = 3$, i.e., $S' = \{o, o'_1, o'_2\}$ and $a = 2$, it could be verified that S_o involves exactly three objects (this is because (1) S_o involves no more than three objects by definition and (2) S_o involves at least three objects since otherwise S' involves less than three objects which leads to a contradiction) and thus $cost(S_o) = d(o, q) \cdot 3 = cost(S')$. Still, S' is an exact solution. In the case that $|S'| \geq 4$, i.e., $a \geq 3$, we continue to prove as follows. First, it could be verified that $b \geq 2$. Second, it could be verified that $|\psi_1| \geq 3$ since $a \geq 3$. Third, in the case that $|\psi_1| \in [3, 7]$, we verify that $\frac{cost(S')}{cost(S_o)} \leq \ln |q.\psi|$ and the details of this step could be found in [6]. Therefore, in the following, we focus on the case that $|\psi_1| \geq 8$.

First, we have

$$|o'_t \cap \psi_t| \geq \frac{|\psi_t|}{b} \tag{2}$$

which could be verified by as follows. Consider the moment when o'_{t-1} (if any) has been included in S' but o'_t has not. By the greedy nature of the process, o'_t is the object that covers the greatest number of keywords that are not covered yet, i.e., $o'_t = \arg\max_{o \in \mathcal{O}'} \{|o.\psi \cap \psi_t|\}$. Therefore, we know $|o'_t.\psi \cap \psi_t| \geq \max_{1 \leq i \leq b} \{|o_i.\psi \cap \psi_t|\} \geq \frac{\sum_{1 \leq i \leq b} |o_i.\psi \cap \psi_t|}{b} \geq \frac{|\psi_t|}{b}$.

Based on Equation (2), we have

$$\begin{aligned} |\psi_{t+1}| &= |\psi_t| - |o'_t.\psi \cap \psi_t| \\ &\leq |\psi_t| - \frac{|\psi_t|}{b} = (1 - \frac{1}{b})|\psi_t| \end{aligned} \tag{3}$$

Based on Equation (3), we further deduce that

$$|\psi_a| \leq (1 - \frac{1}{b})|\psi_{a-1}| \leq (1 - \frac{1}{b})^{a-1}|\psi_1| \tag{4}$$

Note that $|\psi_a| \geq 1$ since otherwise o'_a would not be included in S'. As a result, we know

$$(1 - \frac{1}{b})^{a-1}|\psi_1| \geq |\psi_a| \geq 1 \tag{5}$$

Based on Equation (5), we know that

$$a \leq \frac{\ln|\psi_1|}{-\ln(1 - \frac{1}{b})} + 1 \leq \frac{\ln|\psi_1|}{\frac{1}{b}} + 1 \tag{6}$$

$$\leq \ln|\psi_1| \cdot b + 1 \tag{7}$$

The correctness of Equation (6) is based on the fact that $\ln(1 + x) \leq x$ for $x \in (-1, 0]$ (to illustrate, consider $f(x) = \ln(1 + x) - x$. We have $f'(x) = \frac{1}{1+x} - 1 \geq 0$ for $x \in (-1, 0]$ and $f(0) = 0$).

As a result, we know

$$\begin{aligned} \frac{cost(S')}{cost(S_o)} &= \frac{d(o, q) \cdot |S'|}{d(o, q) \cdot |S_o|} = \frac{a + 1}{b + 1} \\ &\leq \frac{\ln|\psi_1| \cdot b + 2}{b + 1} \leq \ln|\psi_1| + \frac{2 - \ln|\psi_1|}{b + 1} \\ &\leq \ln|\psi_1| + \frac{2 - \ln 8}{b + 1} < \ln|q.\psi| \end{aligned} \tag{8}$$

which immediately implies the correctness of the theorem since $cost(S_a) \leq cost(S')$. □

Time Complexity. We use the IR-tree built on $\mathcal{O}$ to support the range query operations involved in the algorithm.

Let n_1 be the number of iterations in *MaxDotSize-A*. The time complexity of *MaxDotSize-A* is $O(n_1 \cdot \gamma)$, where γ is the cost of the greedy process. The cost of the greedy process is $O(|\psi| \cdot |\psi||\mathcal{O}'|)$ since it involves at most $|\psi|$ iterations and for each iteration, it checks for at most $|\mathcal{O}'|$ objects the number of keywords in ψ newly covered by the object being checked, which could be done in $O(|\psi| \cdot |\mathcal{O}'|)$ time with the help of an inverted list of $|\mathcal{O}'|$ based on ψ (note that the cost of building the inverted list is simply $O(\sum_{o \in \mathcal{O}'} |o.\psi|)$).

Therefore, the time complexity of *MaxDotSize-A* is $O(n_1 \cdot (\sum_{o \in \mathcal{O}'} |o.\psi| + |\psi|^2|\mathcal{O}'|))$, where $n_1 << |\mathcal{O}_q|$, $|\mathcal{O}'| \leq |\mathcal{O}_q| << |\mathcal{O}|$, and $|\psi| < |q.\psi|$.

4.3 Adaptations of Existing Algorithms

In this section, we adapt the existing algorithms in [3,18,2], which are originally designed for CoSKQ problem with other cost functions, for MaxDotSize-CoSKQ.

Cao-E. *Cao-E* is an exact algorithm proposed in [3] for CoSKQ problem with $cost_{MaxSum}$. It can be adapted to MaxDotSize-CoSKQ problem directly by replacing the cost function from $cost_{MaxSum}$ to $cost_{MaxDotSize}$, because it is a best-first search algorithm which is independent of the cost function used in the problem.

Other Exact Algorithms. Some other exact algorithms were proposed in the literature for CoSKQ problem with different cost functions, namely *Cao-Sum-E* [3,2] (for $cost_{Sum}$), *Cao-E-New* [2] (for either $cost_{MaxSum}$ or $cost_{MinMax}$) and *Long-E* [18] (for either $cost_{MaxSum}$ or $cost_{Dia}$). They cannot be adapted to CoSKQ problem with $cost_{MaxDotSize}$ because they all rely on the property of their original cost functions. Consider *Long-E* as an example. The core of *Long-E* relies on an important property of the distance owner group (containing three objects) that different sets of objects with the same distance owner group have the same cost for the cost function of either $cost_{MaxSum}$ or $cost_{Dia}$ studied in [18]. However, this important property could not be applied to our cost function studied in this paper, i.e., $cost_{MaxDotSize}$. In fact, it is possible that two sets of objects with the same distance owner group have different costs for the cost function of $cost_{MaxDotSize}$.

Cao-A1. *Cao-A1* is an approximate algorithm proposed in [3,2] for CoSKQ problem with either $cost_{MaxSum}$ or $cost_{MinMax}$. In [6], we prove that *Cao-A1* gives $|q.\psi|$-factor approximation for MaxDotSize-CoSKQ.

Cao-A2. *Cao-A2* is an approximate algorithm proposed in [2] for CoSKQ problem with $cost_{MaxSum}$. In [6], we prove that *Cao-A2* gives $\frac{|q.\psi|}{2}$-factor approximation for MaxDotSize-CoSKQ.

Cao-A3. *Cao-A3* is an approximate algorithm proposed in [3,2] for CoSKQ problem with $cost_{Sum}$. In [6], we prove that *Cao-A3* gives $|q.\psi|$-factor approximation for MaxDotSize-CoSKQ.

Long-A. *Long-A* is an approximate algorithm proposed in [18] for CoSKQ problem with either $cost_{MaxSum}$ or $cost_{Dia}$. In [6], we prove that *Long-A* gives $\frac{|q.\psi|}{2}$-factor approximation for MaxDotSize-CoSKQ.

Approximate algorithm	Approximation factor
Cao-A1 [3,2]	$\|q.\psi\|$
Cao-A2 [2]	$\|q.\psi\|/2$
Cao-A3 [3,2]	$\|q.\psi\|$
Long-A [18]	$\|q.\psi\|/2$
MaxDotSize-A (this paper)	$\ln \|q.\psi\|$

Table 1. Approximation factors for MaxDotSize-CoSKQ

Table 1 shows the approximation factors of the above adaptations of existing approximate algorithms and also the approximate algorithm *MaxDotSize-A* in this paper. Among all approximate algorithms, our *MaxDotSize-A* provides the best approximation factor for MaxDotSize-CoSKQ.

4.4 Extension to Arbitrary Inherent Costs

Our algorithms can also be applied to the general case with arbitrary object costs, with the following small changes in both *MaxDotSize-E* and *MaxDotSize-A*.

Specifically, for *MaxDotSize-E*, we do not return the solution immediately after we found a set S'' covers ψ (i.e., line 9 in Algorithm 2). Instead, we enumerate all subsets and find the one with the minimum cost. Note that the distance constraint pruning (Lemma 2) is still applicable. Also, we adjust the definition of dominance as follows. Given a set of keywords ψ, a set of objects $\mathcal{O}'$, two objects o_1 and o_2 in $\mathcal{O}'$, we say that o_1 dominates o_2 wrt ψ if all keywords in ψ that are covered by o_2 can be covered by o_1, there exists a keyword in ψ that is covered by o_1 but not by o_2, and $o_1.cost \leq o_2.cost$. Then, the pruning based on dominance remains applicable. It is easy to see that the above changes do not affect the correctness and time complexity of the algorithm.

For *MaxDotSize-A*, we need to change the selection criteria in the greedy process as follows. The greedy process first initializes the set to be returned to $\{o\}$ and then iteratively selects an object in the disk $D(q, d(o, q))$ which has the greatest ratio of (number of uncovered keywords covered by the object) / (object inherent cost) and inserts it into the set until all keywords are covered. The following theorem shows the approximation ratio of *MaxDotSize-A*.

Theorem 3. *MaxDotSize-A gives a* $(\ln |q.\psi| + 1)$*-factor approximation for the MaxDotSize-CoSKQ problem, when objects have arbitrary inherent costs. In particular, if the solution returned by MaxDotSize-A has the size at most 2, the solution is an exact solution.*

Proof: Let $c = o.cost$, $a = \sum_{o \in S'} o.cost - c$ and $b = \sum_{o \in S_o} o.cost - c$. Let ψ_t denote the set of keywords in $q.\psi$ that are not covered by S' after o'_{t-1} (if any) is included in S' but before o'_t is included in S' for $t = 1, 2, ...$ For example, $\psi_1 = q.\psi - o.\psi$.

In the case that $|S'| = 1$, i.e., $S' = \{o\}$, we know that o covers all the keywords in $q.\psi$ and $S_o = \{o\} = S'$. Therefore, S' is an exact solution. In the case that $|S'| = 2$, i.e., $S' = \{o, o'_1\}$, S_o also involves exactly o and o'_1 and thus $cost(S_o) = cost(S')$. Again, S' is an exact solution.

In the case that $|S'| \geq 3$, we verify $a \leq b \cdot (\ln |\psi_1| + 1)$ as follows. Consider an object $o_i \in S_o$. Let $o_i.\psi = \{t_k, t_{k-1}, ..., t_1\}$, where the algorithm covers the keywords in o_i in the order $t_k, t_{k-1}, ..., t_1$. There exist an iteration in our algorithm that pick an object o'_t that covers a keyword t_j of o_i. In that iteration, at least i keywords in o_i remain uncovered. Thus, if the algorithm were to pick o_i in that iteration, the cost per keyword at most $o_i.cost/i$. Summing over the keywords in o_i, the total amount charged to keywords in o_i is at most $o_i.cost \cdot (1 + 1/2 + 1/3 + ..1/k) \leq o_i.cost \cdot (\ln k + 1) \leq o_i.cost \cdot (\ln |\psi_1| + 1)$. Summing over the objects in S_o and noting that every keywords

in ψ_1 is covered by some objects in S_o, we get

$$a = \sum_{o_i \in S_o \setminus o} o_i.cost \cdot (\ln |\psi_1| + 1)$$
$$= b \cdot (\ln |\psi_1| + 1) \quad (9)$$

Therefore, we know

$$\frac{cost(S')}{cost(S_o)} = \frac{d(o,q) \cdot \sum_{o \in S'} o.cost}{d(o,q) \cdot \sum_{o \in S_o} o.cost} = \frac{a+c}{b+c}$$
$$\leq \frac{b \cdot (\ln |\psi_1| + 1) + c}{b+c}$$
$$\leq \ln |\psi_1| + 1 - \frac{c \cdot (\ln |\psi_1| + 1)}{b+c} < \ln |q.\psi| + 1 \quad (10)$$

which immediately implies the correctness of the theorem since $cost(S_a) \leq cost(S')$. □

It is easy to see that changing the object selection criteria does not affect the time complexity of the algorithm.

5 EMPIRICAL STUDIES

5.1 Experimental Set-up

Datasets. Following the existing studies [3,18,2], we used three real datasets in our experiments, namely Hotel, GN and Web. Dataset Hotel contains a set of hotels in the U.S. (www.allstays.com), each of which has a spatial location and a set of words that describe the hotel (e.g., restaurant, pool). Dataset GN was collected from the U.S. Board on Geographic Names (geonames.usgs.gov), where each object has a location and also a set of descriptive keywords (e.g., a geographic name such as valley). Dataset Web was generated by merging two real datasets. One is a spatial dataset called TigerCensus-Block[1], which contains a set of census blocks in Iowa, Kansas, Missouri and Nebraska. The other is WEBSPAM-UK2007[2], which consists of a set of web documents. Table 2 shows the statistics of the datasets. We set the inherent costs of the objects to 1.

Query Generation. Let $\mathcal{O}$ be a dataset of objects. Given an integer k, we generate a query q with k query keywords similarly as [3,18] did. Specifically, to generate $q.\lambda$, we randomly pick a location from the MBR of the objects in $\mathcal{O}$, and to generate $q.\psi$, we first rank all the keywords that are associated with objects in $\mathcal{O}$ in descending order of their frequencies and then randomly pick k keywords in the percentile range of [10, 40]. In this way, each query keyword has a relatively high frequency.

Algorithms. We studied our *MaxDotSize-E*, *MaxDotSize-A* and adapted algorithms as mentioned in Section 4.3. Specifically, we consider two exact algorithms, namely

[1] http://www.rtreeportal.org

[2] http://barcelona.research.yahoo.net/webspam/datasets/uk2007

	Hotel	GN	Web
Number of objects	20,790	1,868,821	579,727
Number of unique words	602	222,409	2,899,175
Number of words	80,645	18,374,228	249,132,883

Table 2. Datasets used in the experiments

MaxDotSize-E and *Cao-E* [3] (the adaption), and five approximate algorithms, namely *MaxDotSize-A*, *Cao-A1* [3,2], *Cao-A2* [2], *Cao-A3* [3,2] and *Long-A* [18].

All algorithms were implemented in C++ and all experiments were conducted on a Linux platform with a 2.66GHz machine and 32GB RAM. The IR-tree index structure is memory resident.

5.2 Experimental Results

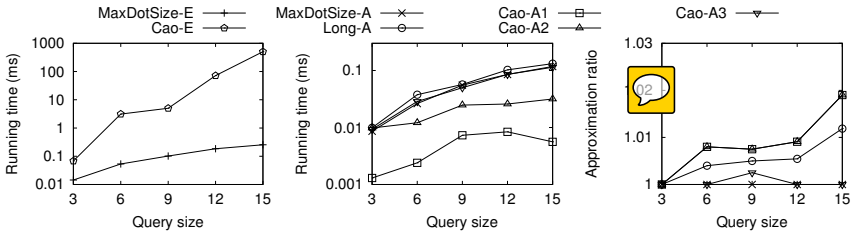
Following the existing studies [3,18,2], we used the running time and the approximation ratio (for approximate algorithms only) as measurements. For each set of settings, we generated 50 queries, ran the algorithms with each of these 50 queries. The averaged measurements are reported.

5.2.1 Effect of $|q.\psi|$

Following the existing studies [3,18], we vary the number of query keywords (i.e., $|q.\psi|$) from $\{3, 6, 9, 12, 15\}$.

Experiment on Dataset Hotel. Figure 2 shows the results on dataset Hotel. According to Figure 2(a), the running time increases with the query size. Our *MaxDotSize-E* is faster than *Cao-E* by 1-3 orders of magnitude, and the order of magnitude increases with the query size. It is because *Cao-E* has to enumerate all sets while *MaxDotSize-E* has more effective pruning strategies. According to Figure 2(b), *MaxDotSize-A*, *Cao-A2*, *Cao-A3* and *Long-A* have comparable running time. The running time of *Cao-A1* is the smallest but as shown in Figure 2(c), however, the empirical approximation ratio of *Cao-A1* is the greatest. Our *MaxDotSize-A* has the best performance with an approximate ratio close to 1, which shows that *MaxDotSize-A* achieves a high accuracy in practice. This is also consistent with our theoretical results that *MaxDotSize-A* gives the best approximation factor among all approximate algorithms.

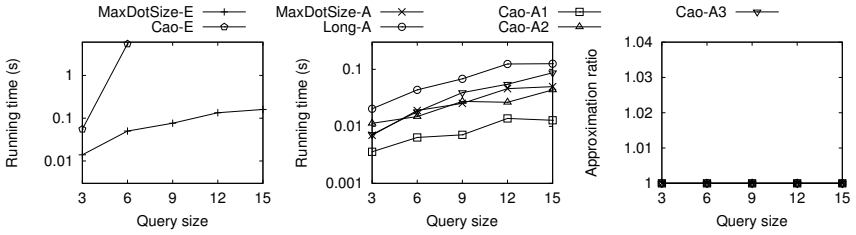
Experiment on Dataset GN. Figure 3 shows the results on dataset GN. The results of *Cao-E* with the query size 9, 12, 15 are not shown because it took more than 3 days or ran out of memory. According to Figure 3(a), our *MaxDotSize-E* is faster than *Cao-E*. Note that when query size increases, the running time of *MaxDotSize-E* only increases slightly because the pruning strategy based on dominant objects reduces the search space effectively. According to Figure 3(b), *Long-A* runs the slowest, *Cao-A1* runs the fastest, and all other approximate algorithms including *MaxDotSize-A* run comparably fast. As shown in Figure 3(c), all approximate algorithms have approximate ratios 1. We found that this is because each object in the optimal solution S_o only contains one



(a) Running time (Exact) (b) Running time (Appro.) (c) Appro. ratio

Fig. 2. Effect of $|q.\psi|$ (Hotel)

query keyword and thus the size of S_o is equivalent to the number of query keywords (i.e., $|S_o| = |q.\psi|$). In this case, $S_o = N(q)$ which is used as a starting point in each of the approximate algorithms.



(a) Running time (Exact) (b) Running time (Appro.) (c) Appro. ratio

Fig. 3. Effect of $|q.\psi|$ (GN)

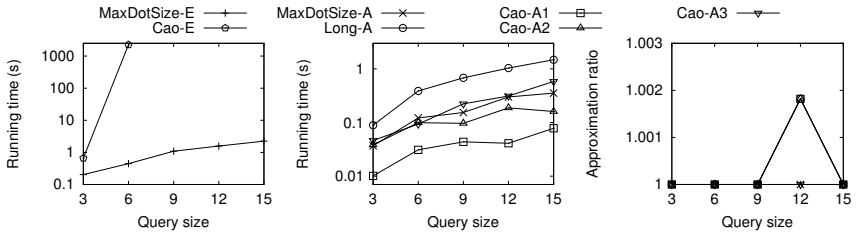
Experiment on Dataset Web. Figure 4 shows the results on dataset Web, which gives similar clues. In the following, we do not show the results of *Cao-E* since it is not scalable and consistently dominated by our *MaxDotSize-E* algorithm.

5.2.2 Effect of average $|o.\psi|$

Following the existing studies [3,18], we conduced experiments on average $|o.\psi|$. The details could be found in [6].

5.2.3 Scalability Test

Following the existing studies [3,18,2], we generated 5 synthetic datasets for the experiments of scalability test, in which the numbers of objects used are 2M, 4M, 6M, 8M and 10M. Specifically, we generated a synthetic dataset by augmenting the GN dataset with additional objects as follows. Each time, we create a new object o with $o.\lambda$ set to be a random location from the original GN dataset by following the distribution and $o.\psi$

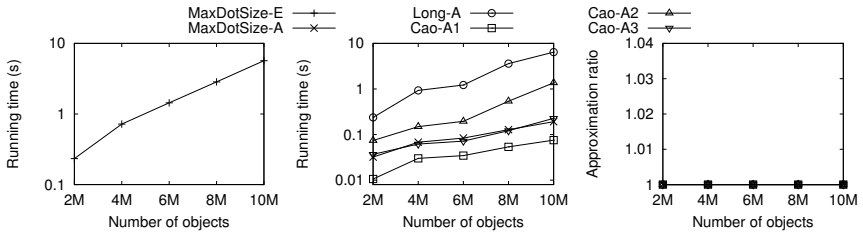


(a) Running time (Exact) (b) Running time (Appro.) (c) Appro. ratio

Fig. 4. Effect of $|q.\psi|$ (Web)

set to be a random document from GN and then add it into the GN dataset. We vary the number of objects from $\{2M, 4M, 6M, 8M, 10M\}$, and the query size $|q.\psi|$ is set to 6.

Figures 5 shows the results for the scalability test. According to Figure 5(a), our *MaxDotSize-E* is scalable wrt the number of objects in the datasets, e.g., it ran within 10 seconds on a dataset with 10M objects. Besides, according to Figure 5(b), our *MaxDotSize-A* runs consistently faster than *Cao-A2* and *Long-A* and it is scalable, e.g., it ran within 1 second on a dataset with 10M objects. According to Figure 5(c), all approximate algorithms can achieve approximation ratios close to 1.



(a) Running time (Exact) (b) Running time (Appro.) (c) Appro. ratio

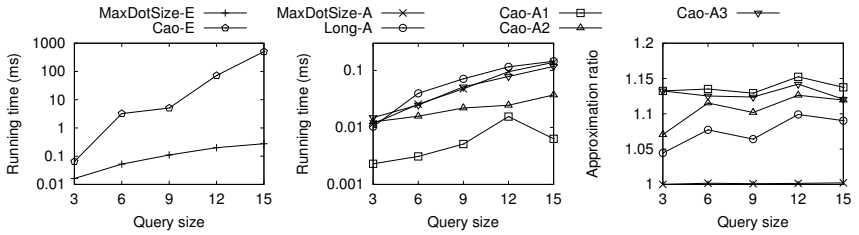
Fig. 5. Scalability test

5.2.4 Objects with Inherent Cost

We further generated a dataset based on the Hotel dataset, where each object is associated with an inherent cost. For each object, we assign an integer inherent cost in the range $[1, 5]$ randomly.

Figure 6 shows the results. As shown in Figure 6(a) and (b), the running time of the algorithms are similar to the case without object inherent cost (i.e., Figure 2(a) and (b)). According to Figure 6(c), the approximation ratio of our *MaxDotSize-A* is near to 1, which shows that the accuracy of *MaxDotSize-A* is high in practice. The approximation ratio showed in the figure corresponds to the average of 50 queries, we found that the

approximation ratio of *MaxDotSize-A* is exactly 1 for most queries (e.g., more than 47). Therefore, the averaged approximation ratio of *MaxDotSize-A* is always near to 1. This is also consistent with our theoretical results that *MaxDotSize-A* gives the best approximation factor among all approximate algorithms.



(a) Running time (Exact) (b) Running time (Appro.) (c) Appro. ratio

Fig. 6. Effect of $|q.\psi|$ (Inherent cost)

6 Conclusion

In this paper, we proposed a new cost function, *maximum dot size cost* for CoSKQ problem. The cost function captures both the distances among objects and the inherent costs of objects. We proved the NP-hardness of MaxDotSize-CoSKQ and designed an exact algorithm and an approximate algorithm with a theoretical error guarantee. Extensive experiments were conducted which verified our theoretical findings.

There are several interesting future research directions. One direction is to penalize objects with too much keywords such that the results would not always favour objects with many keywords (this could be used to make it more difficult to cheat an algorithm for the CoSKQ problem by associating an object with many keywords). It is interesting to see how to make a good balance between objects with many keywords and the number of objects.

References

1. X. Cao, L. Chen, G. Cong, and X. Xiao. Keyword-aware optimal route search. *PVLDB*, 5(11):1136–1147, 2012.
2. X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi. Efficient processing of spatial group keyword queries. *TODS*, 40(2):13, 2015.
3. X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384. ACM, 2011.
4. X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu. Retrieving regions of intersect for user exploration. *PVLDB*, 7(9), 2014.
5. A. Cary, O. Wolfson, and N. Rishe. Efficient and scalable method for processing top-k spatial boolean queries. In *SSDBM*, pages 87–95. Springer, 2010.

6. H. K.-H. Chan, C. Long, and R. C.-W. Wong. Inherent-cost aware collective spatial keyword queries (full version). In `http://www.cse.ust.hk/~khchanak/paper/sstd17-coskq-full.pdf`, 2017.
7. B. Chazelle, R. Cole, F. P. Preparata, and C. Yap. New upper bounds for neighbor searching. *Information and control*, 68(1):105–124, 1986.
8. D.-W. Choi, J. Pei, and X. Lin. Finding the minimum spatial keyword cover. In *ICDE*, pages 685–696. IEEE, 2016.
9. G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.
10. G. Cong, H. Lu, B. C. Ooi, D. Zhang, and M. Zhang. Efficient spatial keyword search in trajectory databases. *Arxiv preprint arXiv:1205.2880*, 2012.
11. K. Deng, X. Li, J. Lu, and X. Zhou. Best keyword cover search. *TKDE*, 27(1):61–73, 2015.
12. J. Fan, G. Li, L. Z. S. Chen, and J. Hu. Seal: Spatio-textual similarity search. *PVLDB*, 5(9):824–835, 2012.
13. I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665. IEEE, 2008.
14. Y. Gao, J. Zhao, B. Zheng, and G. Chen. Efficient collective spatial keyword query processing on road networks. *ITS*, 17(2):469–480, 2016.
15. T. Guo, X. Cao, and G. Cong. Efficient algorithms for answering the m-closest keywords query. In *SIGMOD*. ACM, 2015.
16. Z. Li, K. Lee, B. Zheng, W. Lee, D. Lee, and X. Wang. Ir-tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011.
17. J. Liu, K. Deng, H. Sun, Y. Ge, X. Zhou, and C. Jensen. Clue-based spatio-textual query. *PVLDB*, 10(5):529–540, 2017.
18. C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu. Collective spatial keyword queries:a distance owner-driven approach. In *SIGMOD*, pages 689–700. ACM, 2013.
19. J. Rocha, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. *SSTD*, pages 205–222, 2011.
20. J. B. Rocha-Junior and K. Nørvåg. Top-k spatial keyword queries on road networks. EDBT, pages 168–179. ACM, 2012.
21. S. Shang, R. Ding, B. Yuan, K. Xie, K. Zheng, and P. Kalnis. User oriented trajectory search for trip recommendation. In *EDBT*, pages 156–167. ACM, 2012.
22. A. Skovsgaard and C. S. Jensen. Finding top-k relevant groups of spatial web objects. *VLDBJ*, 24(4):537–555, 2015.
23. S. Su, S. Zhao, X. Cheng, R. Bi, X. Cao, and J. Wang. Group-based collective keyword querying in road networks. *Information Processing Letters*, 118:83–90, 2017.
24. D. Wu, G. Cong, and C. Jensen. A framework for efficient spatial web object retrieval. *VLDBJ*, 21(6):797–822, 2012.
25. D. Wu, M. Yiu, G. Cong, and C. Jensen. Joint top-k spatial keyword query processing. *TKDE*, 24(10):1889–1903, 2012.
26. D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552. IEEE, 2011.
27. Y. Zeng, X. Chen, X. Cao, S. Qin, M. Cavazza, and Y. Xiang. Optimal route search with the coverage of users' preferences. In *IJCAI*, pages 2118–2124, 2015.
28. D. Zhang, Y. M. Chee, A. Mondal, A. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699. IEEE, 2009.
29. D. Zhang, B. C. Ooi, and A. K. H. Tung. Locating mapped resources in web 2.0. In *ICDE*, pages 521–532. IEEE, 2010.
30. D. Zhang, K.-L. Tan, and A. K. H. Tung. Scalable top-k spatial keyword search. EDBT/ICDT, pages 359–370. ACM, 2013.